Proiect Analiza Algoritmilor

Etapa 1

Ilie Dana Maria 324 CA

Facultatea de Automatica si Calculatoare
Universitatea POLITEHNICA Bucuresti

1 Introducere

1.1 Descrierea problemei rezolvate

In matematica, o multime reprezinta o colectie de obiecte bine determinate si distincte. Obiectele constituente ale multimii poarta numele de "elementele multimii". Elementele care formeaza o multime pot fi de orice fel: numere, puncte in spatiu, simboluri, alte multimi etc. Putem spune despre doua multimi ca sunt egale daca sunt formate din aceleasi elemente. O multime fara niciun element se numeste "vida". Conceptul de multime a aparut la sfarsitul secolului al XIX-lea si constituie si in prezent unul dintre cele mai importante subiecte ale matematicii, intelegerea multimilor si a teoriei multimilor fiind o problema de mare interes atat in matematica, cat si in programare - unde pot fi valorificate pentru rezolvarea mai usoara a multor probleme.

In programare, multimile reprezinta tipuri de date abstracte care stocheaza valori unice, fara o ordine particulara, constituind chiar implemetarea multimilor finite din matematica. Intre multimi se pot defini o serie de operatii principale precum: reuniune, intersectie, diferenta s.a.m.d, dar si operatii specifice, dependente de caracterul static(nu se modifica dupa creare) sau dinamic(permite inserarea si stergerea elementelor) al multimii. Cele ce urmeaza se vor axa pe implemetarea multimilor si a operatiilor de adaugare, stergere si modificare a unui element, folosind structuri de date.

1.2 Exemple de aplicatii practice

Multimile au o aplicabilitate practica variata, oferind posibilitatea formalizarii realitatii la nivelul unui program. Un exemplu in acest sens poate fi gruparea angajatilor de la o anumita companie sub forma unei multimi pentru a-i gestiona, sau crearea unei aplicatii care stocheaza melodii in diferite playlisturi, in functie de gen sau preferintele utilizatorului.

1.3 Specificarea solutiilor alese

Multimile pot fi implementate folosind structuri de date diverse, fiecare prezentand atat avantaje, cat si dezavantaje la nivel de timp si spatiu, in functie de operatiile realizate, scopul programului, setul de date de intrare sau de resursele sistemului care executa sarcinile.
Pentru rezolvarea problemei am ales implementarea multimilor cu ajutorul tabelelor de dispersie si al arborilor binari de cautare echilibrati, mai exact Treap-uri.

O tabela de dispersie este o structura de date care stocheaza perechi cheie-valoare, realizand indexarea acestora cu ajutorul unei functii hash. Ideal, functia de hash va asigna fiecarei chei un index unic, dar, de cele mai multe ori, aceasta functie este imperfecta, generand coliziuni care pot fi tratate prin diferite metode(inlatuire sau adresare deschisa).

Un Treap este un arbore binar de cautare echilibrat, in care fiecarui nod ii este asignata o prioritate in mod aleatoriu. Aceasta structura trebuie sa respecte doua proprietati sau invarianti. Prima propritate este cea de arbore binar de cautare, conform careia cheia unui nod este mai mica(sau egala) decat cheia fiului dreapta, daca exista si mai mare(sau egala) decat cheia fiului stanga, daca exista. Cea de-a doua proprietate este cea de heap si se refera la prioritatea unui nod, care trebuie sa fie mai mare(sau egala) decat prioritatile fiilor.

1.4 Criteriile de evaluare pentru solutia propusa

Pentru evaluarea solutiei propuse voi folosi o serie de teste variate care sa verifice atat corectitudinea, cat si eficienta solutiilor propuse. In ceea ce priveste varietatea testelor, voi avea atat seturi de date sortate, cat si ordonate aleator. De asemenea, initial voi testa pe multimi cu mai putine elemente si, ulterior, voi genera si seturi de date cu mai multe elemente. Vor fi teste care sa verifice exclusiv fiecare operatie suportata de structurile de date in parte, dar si teste pentru operatii multiple combinate. Pentru a evalua corectitudinea implementarii algoritmilor voi avea o functie care sa testeze automat caracterul de multime, mai exact sa verifice ca nu exista elemente duplicate. In plus, in cazul treap-ului voi avea, de asemenea, o functie care sa testeze automat atat proprietatea de arbore binar de cautare, cat si cea de heap.

2 Prezentarea solutiilor

2.1 Descrierea modului in care functioneaz algoritmii alesi

2.1.1 Tabela de dispersie

Adaugarea unui element in tabela de dispersie presupune realizarea a doua operatii. Prima operatie consta in transformarea cheii intr-un index intreg, strict pozitiv, prin intermediul unei functii de hash. In mod ideal, chei diferite mapeaza indexuri diferite. Acest lucru nu se intampla insa si in realitate deoarece lungimea hash-ului este fixa, iar obiectele stocate pot prezenta

lungimi si continut arbitrare. De aceea, pentru a fi considerata buna, o functie de hash trebuie sa minimizeze numarul de chei diferite care produc acelasi index (coliziuni). Astfel, cea de a doua operatie din cadrul adaugarii in tabela de dispersie este rezolvare coliziunilor, prin folosirea inlantuirii (direct chaining) sau a adresarii deschise (linear probing). Metoda inlantuirii directe presupune utilizarea listelor inlantuite pentru a stoca in acelasi "bucket" valorile care au produs hash-uri identice. Prin urmare, este folosita cate o lista inlantuita pentru fiecare "bucket"al tabelului de dispersie, avand deci un array de liste. Fiecare lista este asociata unui anumit hash, obtinut prin aplicarea unei functii de hashing asupra cheii.Aceasta metoda este potrivita pentru situatia in in care nu se cunosc dinainte toate operatiile care vor fi aplicate sau numarul aproximativ al lor. Metoda adresarii deschise trateaza coliziunile prin modificarea pozitiei pe care trebuie adaugata cheia la urmatoarea locatie libera disponibila. In implementarea pe care am realizat-o am ales sa tratez coliziunile folosindu-ma de metoda inlantuirii.

Stergerea unui element din tabela de dispersie presupune cautarea si scoaterea acestuia din lista corespunzatoare.

Cautarea unui element in tabela de dispersie consta in determinarea indexului cu ajutorul functiei de hash si iterarea prin lista de la acel index pentru a indentifica pereche cheie-valoare potrivita.

2.1.2 Treap

Adaugarea unui element in Treap se realizeaza prin generarea unei prioritati aleatoare si adaugarea (respectand invariantul arborelui de cautare) nodului la baza arborelui printr-o procedura recursiva, pornind de la radacina. Tipul de date trebuie sa permita o relatie de ordine pentru a facilita compararea elementelor. Astfel, Treap-ului ii este asociata o functie de comparare care intoarce valoarea 0 daca cele doua elemente sunt egale, o valoare mai mica decat 0 daca elementul nou introdus este mai mic si o valoare mai mare decat 0 daca elementul nou introdus este mai mare. De asemenea, pentru mentinerea invariantului de heap sunt aplicate operatii de rotire la stanga sau la dreapta asupra nodului, in cazul in care prioritatea sa este mai mare decat cea a parintelui.

Stergerea unui element din Treap se efectueaza prin rotirea nodului pe care dorim sa il stergem pana cand ajunge la baza arborelui si eliminarea acestuia atunci.

2.2 Analiza complexitatii solutiilor

2.2.1 Tabela de dispersie

Complexitatea temporala in cazul mediu al tabelei de dispersie pentru functiile de cautarea, inserare si stergere este $O(1)$ (timp constant)

In cazul cel mai defavorabil, complexitatea temporala poate fi $O(n)$. Acest lucru se intampla atunci cand, dupa aplicarea hash-ului, toate elementele se vor afla in acelasi "bucket", complexitatea fiind data de parcurgerea listei inlantuite in intregime.

2.2.2 Treap

Complexitatea temporala in cazul mediu al Treap-ului pentru operatiile de cautare, adaugare si stergere este $O(log\ n)$

In cazul cel mai defavorabil, complexitatea temporala a operatiilor de baza este $O(n)$. Acest lucru se intampla atunci cand datele introduse in arbore sunt sortate, arborele transformandu-se intr-o lista, care prezinta complexitatea operatiilor $O(n)$.

2.3 Prezentarea principalelor avantaje si dezavantaje pentru solutiile luate in considerare

2.3.1 Tabela de dispersie

Avantajele tabelelor de dispersie sunt date de simplitatea operatiei de stergere si de posibilitatea amanarii redimensionarii tabelei mult timp, performanta fiind suficient de buna chiar si atunci cand toate pozitiile din hashtable sunt folosite. Tabelele de dispersie sunt eficiente in mod deosebit atunci cand se cunoaste numarul maxim de intrari, astfel incat array-ul de "bucket-uri" sa fie alocat o singura data si sa nu apara necesitatea redimensionarii.

Dezavantajele tabelei de dispersie sunt mostenite de la listele inlantuite si constau in faptul ca, pentru stocarea unor date mici, overhead-ul introdus este semnificativ, iar parcurgere listei este costisitoare. Un alt dezavantaj al tabelei de dispersie este dependenta de performantele functiei de hash.

2.3.2 Treap

Avantajele unui Treap provin de la cele ale unui vector sortat, prezentand in plus inserarea in timp logaritmic. Astfel, Treap-ul suporta parcurgeri ordonate si faciliteaza obtinerea unui element maxim sau minim. Prin urmare, Treap-ul este mai potrivit atunci cand ordinea elementelor este importanta, lucru prezent in implementarea unei multimi pentru care dorim sa stabilim relatii intre elemente.

Dezavantajul Treap-ului provine din faptul ca operatiile de baza in cazul mediu sunt mai costisitoare in comparatie cu implementarea oferita de o tabela de dispersie.

3 Evaluare

3.1 Descrierea modalitatii de construire a setului de teste folosite pentru validare

La construirea setului de teste folosite pentru validare am avut in vedere utilizarea unor intrari variate, avand atat teste generate manual, cat si teste generate automat. Sunt prezente atat seturi de date sortate crescator, cat si ordoante aleator. Testele cuprind atat elemente cu valori mici, cat si elemente cu valori mari. De asemenea, initial testesele sunt realizate pe multimi cu mai putine elemente si, ulterior, sunt generate teste cu mai multe elmente. In total

am construit 20 de teste. Cel mai mic test contine o singura operatie, iar cel mai mare efectueaza 100000 de operatii.

- testele 1, 2 si 4 - testeaza exclusiv operatia de adaugare pentru putine elemente aleatorii
- testul 3 - testeaza exclusiv operatia de adaugare pentru putine elemente sortate crescator
- testele 5, 6 si 7 - testeaza operatiile de adaugare si stergere pentru putine elemente nesortate
- testele 8, 9 si 10 - testeaza operatiile de adaugare, stergere si modificare pentru putine elemente
- testul 11 - contine doar operatii de stergere pe multimi goale
- testele 12, 13, 14 - testeaza operatiile pe mai multe elemente
- testele 15, 19 - verifica operatiile pentru seturi de date mai mari aleatorii
- testele 16 si 17 - verifica operatiile de adaugare si stergere pentru seturi de date mai mari sortate (1000 operatii)
- testul 18 - verifica operatiile de adaugare si modificare pentru seturi de date mari sortate(2000 operatii)
- testul 20 - contine doar operatii de adaugare a elementelor sortate (100000 operatii)

3.2 Specificatiile sistemului de calcul pe care au fost rualte testele

Toate testele au fost rulate pe statia personala, care are urmatoarele specificatii:

- Procesor: AMD Ryzen 5 5600X @3.90 GHz
- NVIDIA GeForce RTX 3060 Ti 8GB GDDR6
- Memorie RAM: 16GB
- Sistem de operare: Windows 10 Pro

3.3 Ilustrarea folosind grafice/tabele, a rezultatelor evaluarii solutiilor pe setul de teste

In urmatorul tabel este ilustrat timpul de rulare in microsecunde pentru fiecare test in cazul tabelei de dispersie cu o inaltime maxima a array-ului de "bucket-uri" de 100.

Nr. Test	Nr. Elemente	Timp tabela dispersie
1	1	33
2	4	32
3	11	35
4	6	33
5	7	33
6	6	32
7	3	32
8	9	34
9	12	36
10	10	35
11	20	35
12	31	44
13	40	49
14	82	67
15	5026	6087
16	1000	651
17	2000	1190
18	2000	1843
19	450	227
20	1000000	2148119

In urmatorul tabel este ilustrat timpul de rulare in microsecunde pentru fiecare test in cazul Treap-ului

Nr. Test	Nr. Elemente	Timp Treap
1	1	27
2	4	26
3	11	32
4	6	29
5	7	29
6	6	27
7	3	28
8	9	31
9	12	31
10	10	31
11	20	31
12	31	37
13	40	42
14	82	60
15	5026	4856
16	1000	404
17	2000	962
18	2000	1295
19	450	203
20	1000000	236959

3.4 Prezentarea valorilor obtinute pe teste

In urma reprezentarii in tabele a timpilor de rulare, se observa ca pentru primele teste, care contin destul de putine elemente, diferenta de performanta dintre cele doua implementari de multimi nu este semnificativa, valorile timpilor de rulare fiind foarte apropiate. Pentru testele cu mai multe elemente, sunt prezenti timpi de rulare mai buni in cazul implementarii multimii cu ajutorul unui Treap. Aceste rezultate pot fi influentate de performanta functiei de hash (frecventa producerii de coliziuni) si de inaltimea maxima initiala a array-ului de "bucket-uri".

Valorile timpilor de rulare pentru Treap nu cuprind si verificarea corectitudinii, mai exact a celor doi invarianti. In cazul in care se testeaza si acest aspect prin parcurgerea inordine si obtinerea unei sortari crescatoare se obtin urmatoarele valori:

Nr. Test	Nr. Elemente	Timp Treap
1	1	97
2	4	87
3	11	113
4	6	115
5	7	81
6	6	29
7	3	80
8	9	82
9	12	87
10	10	83
11	20	31
12	31	91
13	40	96
14	82	123
15	5026	7326
16	1000	622
17	2000	2066
18	2000	1325
19	450	324
20	1000000	266562

4 Concluzii

In concluzie, in urma unei analize detaliate a celor doua implementari ale unei multimi, cu ajutorul unei tabele de dispersie sau a unui Treap, se poate spune ca exista atat avantaje, cat si dezavantaje in cazul fiecarei solutii de rezolvare. Astfel, alegerea unei metode de implementare depinde de scopul programului, operatiile cele mai frecvent efectuate, tipul setului de date de intrare sau ordinea de introducere a datelor.

In general, tabela de dispersie reprezinta o alternativa mai eficienta, lucru sugerat chiar de complexitatea temporala in cazul mediu pentru operatiile de cautare, inserare si stergere: $O(1)$, comparativ cu $O(log\ n)$ in cazul Treap-ului. Cu toate acestea, performanta mai buna a tabelei de dispersie nu este garantata deoarece, in cazul in care nu se cunosc dinainte operatiile efectuate si mai ales nici un numar aproximativ al lor, operatiile de redimensionare a tabelei se pot dovedi destul de costisitoare (in functie de fecventa acestora).Totodata, atunci cand factorul de incarcare al unei tabele de dispersie este mic, se iroseste un spatiu considerabil. Aceste lucruri pot face implementare multimii cu ajutorul unui Treap o varianta mai buna, neavand aceste limitari.

Un alt lucru important de luat in considerare atunci cand se alege o implemenatare, este scopul programului. In cazul, in care ordinea elementelor in multime este importanta, este preferata folosirea unui Treap pentru stocarea datelor si efectuarea operatiilor intrucat aceasta metoda de abordare a problemei permite obtinerea elementelor sortate in ordine crescatoare printr-o simpla

parcurgere "inordine". De asemenea, invariantul de Heap al Treap-ului permite calcularea elementului maxim sau minim din multime. Aceste operatii nu sunt suportate in mod natural si in cazul tabelei de dispersie. In cazul in care cele precizate mai sus nu sunt relevante pentru implementarea aleasa sau in cazul in care elementele multimii nu suporta o relatie de ordine bine definita, este preferata utilizarea unei tabele de dispersie.

Bibliografie

1. https://ocw.cs.pub.ro/courses/sd-ca/laboratoare/lab-10

2. https://ocw.cs.pub.ro/courses/sd-ca/laboratoare/lab-04

3. Robert R. Stoll, Set Theory and Logic

4. https://isaaccomputerscience.org/concepts/data_numsys_sets?examBoard=all&stage=all